Random Numbers for Parallel Computations(The 7th Workshop on Stochastic Numerics)

AUTHOR(S):

Fushimi, Masanori

CITATION:

Fushimi, Masanori. Random Numbers for Parallel Computations(The 7th Workshop on Stochastic Numerics). 数理解析研究所講究録 2006, 1462: 57-62

ISSUE DATE:

2006-01

URL:

http://hdl.handle.net/2433/47989

数理解析研究所講究録 1462 巻 2006 年 57-62

Random Numbers for Parallel Computations

Masanori Fushimi

Department of Mathematical Sciences
Nanzan University
27 Seirei-cho, Seto, Aichi 489-0863, Japan
fushimi@nanzan-u.ac.jp

1. Introduction

Large scale Monte Carlo simulations need lots of computational time. A method of obtaining computational results quickly is a parallel computation using many personal computers (PC's). If we want to perform N replications in a simulation and can use m PC's, we may assign about N/m replications to each PC and collect the computational results of m PC's to obtain the final answer as a result of N replications.

In this case, it is important that we use mutually independent or at least uncorrelated random numbers on each PC. If we can use truly random numbers on each PC, there will be no problem. In most Monte Carlo simulations, however, we use pseudorandom numbers instead of truly random numbers, and this may cause some problems. If we use different pseudorandom number generators on each PC, we need to verify the mutual independence or no correlations among the sequences to be generated by these generators, but this is an extremely difficult problem, and there have been few researches on this problem. Under these circumstances, the almost only solution is to choose one generator whose autocorrelation function is known and use mutually uncorrelated subsequences on each PC.

Many pseudorandom number generators have been

proposed so far, and the following generators are specified in Japanese Industrial Standard JIS Z 9031 : 2001[1].

- Linear congruential method
- 3-term GFSR generators
- 5-term GFSR generators
- Combined Tausworthe generators
- Mersenne Twister
- Irrational rotation method

It is very easy to use linear congruential generators for parallel computations, but the periods of these generators are rather short for large scale simulations. The autocorrelation functions of the sequences generated by combined Tausworthe generators, Mersenne Twister, and the irrational rotation method are unknown so that they are not suitable for parallel computations at present. So we consider using GFSR generators for parallel computations in this paper because they have extremely long periods enough for large scale simulations and their auto-correlation functions are known.

2. Characteristics of GFSR generators

A GFSR generator uses the following recursion to generate a sequence of integers $\{X_t\}$.

$$X_{t+p} = X_{t+q_1} \oplus X_{t+q_2} \oplus \cdots \oplus X_{t+q_l} \oplus X_t \tag{1}$$

Here, the symbol $\oplus$ means the bitwise addition modulo 2, and the characteristic polynomial of the recursion (1)

$$f(z) = z^p - z^{q_1} - z^{q_2} - \cdots - z^{q_l} - 1 \tag{2}$$

is a primitive polynomial of degree p over GF(2). In order to use the recursion, we must give initial values $X_1, X_2, \cdots, X_p$, and an initialization method is described in the appendix of [1] based on the method proposed by Fushimi [2]. This initialization method guarantees a good autocorrelation property as well as a good multidimensional property. The autocorrelation function of the sequence generated with this initialization method is given

below as follows [2].

Let $\{X_t\}$ be the b-bit integer sequence generated by the recursion (1) and $\{x_t\} = \{X_t/2^b\}$ be the normalized sequence. The periods of these sequences are $T = 2^p - 1$. Let $\bar{x}$ denote the average of x_t's over the entire period, which is equal to $0.5(1-2^{-b})/(1-2^{-p})$ and very close to 0.5 if p is large. Let $R(s)$ be the autocorrelation function of the sequence $\{x_t\}$ defined by the following.

$$R(s) = \frac{1}{T}\sum_{t=0}^{T-1}(x_t - \bar{x})(x_{t+s} - \bar{x}) \tag{3}$$

It is shown in [2] that

$$R(s) = -\frac{1}{4T}(1-2^{-b})^2 \quad (1 \le |s| \le s_0), \tag{4}$$

which is almost equal to zero because T is very large. Here, $s_0 = (T+1)/b$ if b is a power of 2, e.g. $b = 32$, and in general $s_0 = (T+1)/b'$, where b' is a smallest power of 2 which is not less than b. Thus s_0 is also very large if, for example, $p = 521$ and $b = 32$.

3. Using GFSR generators for parallel computations

As we mentioned before, we use one and the same generator for a simulation, and use mutually uncorrelated subsequences on each PC. More precisely, we choose an integer τ which is less than or equal to s_0/m, where m is the number of PC's we use, and use the subsequences $\{X_t : (k-1)\tau + 1 \le t \le k\tau\}$ on the k-th PC $(k = 1, 2, \cdots, m)$. Then we must compute "initial values" for PC's no. 2 through no. m from the initial values $X_0^{(1)} = \{X_1, X_2, \cdots, X_p\}$ for PC no.1. To compute these initial values quickly when $X_0^{(1)}$ is given, we make preparations for initialization before $X_0^{(1)}$ is specified.

It is known, see e.g. [2] or [3], that the sequence $\{X_t\}$ which is generated by the recursion (1) satisfies

$$X_t = X_{t-e(p-q_1)} \oplus X_{t-e(p-q_2)} \oplus \cdots \oplus X_{t-ep} \tag{5}$$

where e is any integral power of 2. Using the recursion (5), we can compute X_t from $X_0^{(1)}$ rather quickly even for a very large t.

Let $g(u)$ denote the greatest integral power of 2 which does not exceed u. Let $e = g(t/p)$ in (5), then X_t can be computed from $X_{t'}$'s on the right-hand side with relatively small indices t'. We apply the same technique to those $X_{t'}$'s, and repeat this process until X_t is expressed as a linear combination (in the sense of $\oplus$) of the elements of $X_0^{(1)}$.

We illustrate this procedure with a simple example: $p = 521$, $l = 1$, $q = 32$, and $t = 52101$. In this case, we have $g(t/p) = g(52101/521) = g(100) = 64$, and

$$\begin{aligned} X_{52101} &= X_{52101-64\cdot 489} \oplus X_{52101-64\cdot 521} \\ &= X_{20805} \oplus X_{18757}. \end{aligned}$$

Next, since $g(20805/521) = 32$ and $g(18757/521) = 32$, we have

$$\begin{aligned} X_{20805} &= X_{20805-32\cdot 489} \oplus X_{20805-32\cdot 521} \\ &= X_{5157} \oplus X_{4133} \end{aligned}$$

$$\begin{aligned} X_{18757} &= X_{18757-32\cdot 489} \oplus X_{18757-32\cdot 521} \\ &= X_{3109} \oplus X_{2085}. \end{aligned}$$

Repeating the similar computations, we finally obtain the following expression.

$$\begin{aligned} X_{52101} = X_1 \oplus X_{15} \oplus X_{29} \oplus X_{47} \oplus X_{111} \oplus X_{129} \oplus X_{175} \oplus X_{203} \\ \oplus X_{221} \oplus X_{267} \oplus X_{468} \oplus X_{486} \oplus X_{500} \oplus X_{504} \oplus X_{518} \end{aligned}$$

The above procedure can easily be implemented if we use a programming language which provides a function of recursive calls.

If we want to speed up further the above initialization procedure, we may choose τ so that the following equality holds.

$$\tau + 1 - g((\tau+1)/p)p = 1 \tag{6}$$

We illustrate a choice of τ for the case $p = 521$, $l = 1$, $q_1 = 32$, $b = 32$, and $m = 100$. In this case, $T = 2^{521} - 1$, $s_0 = (T+1)/b = 2^{489}$, $\tau \le s_0/m = 2^{489}/100$, $\tau/p \le 2^{489}/(100 \cdot 521) \le 2^{489}/2^{15} = 2^{474}$. So, if we choose $\tau = 2^{474} \cdot 521$, then $e = g((\tau+1)/p) = 2^{474}$, and the equality (6) holds. Then the initial value $X_{\tau+1}$ for PC no. 2 can be expressed using (5) as follows.

$$X_{\tau+1} = X_{\tau+1-489e} \oplus X_{\tau+1-521e}$$
$$= X_{1+32e} \quad \oplus X_1$$

Next we repeatedly apply the recursion (5) to X_{1+32e} in order to express it as a linear combination (in the sense of $\oplus$) of the elements of $X_0^{(1)}$, the initial values for PC no. 1.

Once $X_{\tau+1}$ is expressed as a linear combination of the elements of $X_0^{(1)}$, the other expressions for the initial values $X_{\tau+2}, X_{\tau+3}, \cdots, X_{\tau+p}$ for PC no. 2 can be obtained with slight modifications of the expression for $X_{\tau+1}$. It is important to note that these expressions are independent of the actual numerical values given to $X_0^{(1)}$ in a particular simulation, so that it is enough to compute them only once when the recursion (1) and τ are given. Actual initial values for PC no. 2 are computed using these expressions when $X_0^{(1)}$ is given numerically in a particular simulation. Similarly, the initial values for PC no. k can be computed from the initial values for PC no. $(k-1)$, $3 \le k \le m$.

The number of pseudorandom numbers assigned to one PC is τ, and it is equal to $2^{474} \cdot 521 \simeq 10^{145}$ in the above example, which is extremely too many to use in one simulation. So, we can choose a much smaller number for τ satisfying (6), which will greatly reduce the time required for initialization. For example, if we choose $\tau = 2^{60} \cdot 521 \simeq 10^{20}$, it will be more than enough for any realistic simulations.

4. Conclusion and final remarks

We have shown a method for using GFSR pseudorandom numbers for parallel computations. It is recommended to choose a number $\tau = 2^e \cdot p$, where e is a positive integer and p is the degree of the characteristic primitive polynomial of the GFSR generator we use, such that τ is larger than the number of pseudorandom numbers to be used on each PC and less than $2^p / bm$, where b is the bit-length of the random numbers to be generated and m is the number of PC's to be used in parallel. A method of computing initial values for each PC is described. The random numbers generated on each PC are uncorrelated.

It is desirable to find methods of using other pseudorandom

number generators, e.g. irrational rotation method, Mersenne Twister and combined Tausworthe generator, for parallel computations.

References

[1] Japanese Industrial Standards Committee (2001) : JIS Z 9031:2001 Procedure for random number generation and randomization (in Japanese), Japanese Standards Association, Tokyo.

[2] Fushimi, M. (1989) : Random Numbers (in Japanese), University of Tokyo Press, Tokyo.

[3] Golomb, S. W. (1982) : Shift Register Sequences, 2nd ed., Aegean Part Press, Laguna Hills, California.